# AI2Bh Planning

Michael Fourman
Revised by Alan Bundy

May 12, 2004

*In which we model action and change. Actions are operations that take us from one situation to another. We start in some initial situation, with the goal of reaching another situation with some desirable properties. A plan is a sequence of actions that will take us from the initial situation to a goal situation.*

## Blocks World

We use examples from the Blocks World. A number of blocks (imagine they are wooden cubes) may sit on the table, or be stacked in towers, one on another, or be held in the hand. (There is only one hand.) A situation is a particular configuration of blocks, some on the table, some possibly stacked on others, and possibly one in the hand.

A configuration can be specified by giving the truth values of a number of predicates:

| | |
|---|---|
| `ontable(a)` | Is block `a` on the table? |
| `on(a,b)` | Is block `a` on block `b`? |
| `handempty` | Is the hand empty? |
| `holding(a)` | Is the hand holding block `a`? |

We say a block is `clear` if it is not in the hand, and no other block is stacked on top of it. The possible actions, and the circumstances in which they can be performed are:

| | |
|---|---|
| `pickup(a)` | Pickup `a`, which is clear and on the table, in the hand, which is empty. |
| `putdown(a)` | Put `a`, which is in the hand, on the table. |
| `stack(a,b)` | Place `a`, which is in the hand, on `b`, which is clear. |
| `unstack(a,b)` | Pick up `a`, which is clear, and stacked on `b`, in the hand, which is empty. |

After performing an action we can observe its effects:

| | |
|---|---|
| `pickup(a)` | Afterwards, the hand is holding `a`, which is neither clear nor on the table. |
| `putdown(a)` | Afterwards, the hand is empty, a is clear, and on the table. |
| `stack(a,b)` | Afterwards, `a`, which is clear, and not in the hand, is on `b`, which is not clear; the hand is empty. |
| `unstack(a,b)` | Afterwards, `a`, which is not clear, is in the hand, which is not empty, and not on `b`, which is clear. |

# Situation Calculus

The Situation Calculus uses first-order predicate logic to reason about situations and actions. In the situation calculus, we *reify* situations and actions, that is we treat them as objects we can reason about. This means that, as well as representing the objects, functions and relations of the world, using constants, functions and relations, we *introduce types to represent situations and actions*, together with a fundamental operation

$$\texttt{Result} : \texttt{Action} \times \texttt{Situation} \longrightarrow \texttt{Situation}$$

The result of performing an action in a given situation is a new situation.

In order to express the assertion that some relation holds in a given situation, we add a situation parameter to each atomic assertion about the world. For example, in place of `on(a,b)`, we use `on(a,b,s)` meaning, `a` is stacked on `b`, in situation `s`.

If $\varphi$ is a logical formula expressing some assertion about the world, in the language without situation variables, we write $\varphi(s)$ for the result of adding the situation parameter, $s$, to each atomic formula occurring in $\varphi$. This gives the formula in situation calculus that expresses $\varphi$ holds in situation $s$.

For example, if the formula, `goal`, is true only in goal situations, then the situation calculus formula $\texttt{goal}(s)$ expresses the fact that $s$ is a goal situation.

## Actions

In the situation calculus, we define an action by axiomatising the relationship between an original situation and the situation that results from performing the action.

**Effect axioms** Each action is axiomatised: the result of performing an action, $A$, in a situation satisfying $A$'s preconditions, is a situation including $A$'s effects.

$$\begin{aligned} & \texttt{ontable}(x,s) \\ \wedge\ & \texttt{clear}(x,s) \\ \wedge\ & \texttt{handempty}(s) \\ & \rightarrow \\ & \quad \texttt{holding}(x,\texttt{Result}(\texttt{pickup}(x),s)) \\ & \wedge\ \neg\texttt{clear}(x,\texttt{Result}(\texttt{pickup}(x),s)) \\ & \wedge\ \neg\texttt{handempty}(\texttt{Result}(\texttt{pickup}(x),s)) \\ & \wedge\ \neg\texttt{ontable}(x,\texttt{Result}(\texttt{pickup}(x),s)) \end{aligned}$$

**Frame axioms** In addition to specifying an action's effects, saying which fluents it changes, we also need to stipulate that it does not affect other fluents. It is simpler to list all the actions that *can* change a given fluent, than to exhaustively enumerate all the fluents that are unchanged by each action. For example, no action other than $\texttt{pickup}(x)$ can remove a block, $x$, from the table:

$$\texttt{ontable}(x,s) \wedge a \neq \texttt{pickup}(x) \rightarrow \texttt{ontable}(x,\texttt{Result}(a,s)$$

## Goal-directed reasoning

The planning problem can be formalised in the situation calculus. Given predicates, `init`, and `goal`, specifying initial and goal states, we postulate an initial situation, $s_0$, such that $\texttt{init}(s_0)$, and search for a term $s$ satisfying $\texttt{goal}(s)$.

# STRIPS

STRIPS is a simple, but enduring, representation for simple planning problems. A situation, also called a *state*, is identified with the values of a finite number of boolean *fluents*. Fluents are boolean state variables whose values can be changed by actions. A state, or situation, is characterised by the set of fluents true in that state. If we start from a predicate calculus representation, each atomic sentence (ie atomic formula without free variables) gives a fluent. For example, in a blocks world with three blocks, `a`, `b`, `c`, we have sixteen fluents: nine of the form $\texttt{on}(x,y)$, three each of the forms $\texttt{ontable}(x)$, and $\texttt{holding}(x)$,and one `handempty`.

Actions are represented as operations that change the set of true fluents, by adding or deleting items. Each action has a precondition — that certain fluents must be true before the action can be performed. In the blocks world, we can represent the pickup action, as follows:

$$\begin{aligned} \texttt{action:pickup}(x) \\ \texttt{precondition:ontable}(x), \texttt{clear}(x), \texttt{handempty} \\ \texttt{add:holding}(x) \\ \texttt{remove:ontable}(x), \texttt{clear}(x), \texttt{handempty} \end{aligned}$$

In this representation, the frame problem is dealt with operationally: unless an action explicitly changes a fluent, it doesn't change.

## Propositional Planning

Formulae of propositional logic, using the fluents as propositional letters, correspond to sets of states.

$$\texttt{on(b,c)} \wedge \texttt{ontable(c)}$$

represents the set of the three states in which `b` is on `c`, which is on the table — `a` may be in the hand, on the table, or on `b`[1].

We consider two propositional representations of the planning problem – corresponding to the STRIPS and situation calculus approaches.

### Propositional STRIPS Planning

We see how to use the propositional representation to construct a formula representing the set of states reachable in $n$ steps from an initial state. First, if $\varphi$ represents some set of states, and $A$ is an action, we construct a Quantified Boolean Formula (QBF)[2], $\varphi\langle A\rangle$, representing the set of states that can be reached from $\varphi$ by performing the action $A$ once.

Take the `pickup(a)` action as an example. The states in $\varphi$ for which the preconditions of the action are satisfied is given by

$$\varphi \wedge \texttt{ontable(a)} \wedge \texttt{clear(a)} \wedge \texttt{handempty}$$

Call these the *activated* states.

The action may change the values of the four propositional variables

$$\texttt{ontable(a)},\ \texttt{holding},\ \texttt{clear(a)},\ \texttt{handempty}.$$

We use existential quantification to give a representation for the set of states that can be obtained from one of the activated states by suitably changing the values of these four propositional variables.

$$\begin{array}{l}\exists\ \texttt{holding},\ \texttt{ontable(a)},\ \texttt{clear(a)},\ \texttt{handempty}.\\ \quad \varphi \wedge \texttt{ontable(a)} \wedge \texttt{clear(a)} \wedge \texttt{handempty}\end{array}$$

Propositionalvariables range over boolean values. So, this is just a shorthand for enumerating the disjunction resulting from substituting in the body,

$$\varphi \wedge \texttt{ontable(a)} \wedge \texttt{clear(a)} \wedge \texttt{handempty}$$

the sixteen possible combinations of values for these four propositional variables.

Finally, we express propositionally the effects of the action

$$\texttt{holding(a)} \wedge\ \neg\texttt{ontable(a)} \wedge\ \neg\texttt{clear(a)} \wedge\ \neg\texttt{handempty}$$

Adding this constraint gives the final expression for $\varphi\langle A\rangle$:

$$\begin{array}{l}\left(\begin{array}{l}\exists\ \texttt{holding},\ \texttt{ontable(a)},\ \texttt{clear(a)},\ \texttt{handempty}.\\ \quad \varphi \wedge \texttt{ontable(a)} \wedge \texttt{clear(a)} \wedge \texttt{handempty}\end{array}\right)\\ \qquad\qquad \wedge\\ \texttt{holding(a)} \wedge\ \neg\texttt{ontable(a)} \wedge\ \neg\texttt{clear(a)} \wedge\ \neg\texttt{handempty}\end{array}$$

[1] Formally, there are other possibilities, but these are not reachable by any sequence of actions from any concretely realisable state. For example, formally, `a` might also be on `c`.

[2] See appendix.

The whole thus represents the states reachable from states in $\varphi$ by performing a single `pickup(a)` action. By performing this computation for each action, and forming the disjunction of the results, we can compute all states reachable from $\varphi$ in one step.

Computationally efficient representations of propositional formulae allow us to develop planning algorithms that iterates this computation of sets of states to implement a form of parallel breadth-first search.

### Propositional representation of situation calculus

Another, highly successful, approach to planning is to code a situation calculus formulation propositionally. We represent the planning problem as the problem of satisfying some complex set of propositional constraints.

Once more, we use the notation of predicate logic, but intend that each atomic sentence be treated as separate propositional letter. We model time as a discrete series of instants, and identify situations with times, $t$. Instead of treating actions as operations on situations, we introduce action predicates, $action(t)$, (one for each fully parametrised action) whose intended meaning is that *action* is performed on the situation at time $t$. Its effects will yield the situation at time $t + 1$.

The idea is to write a set of propositional constraints such that any satisfying valuation provides a solution to the planning problem. Knowing which of the predicates, $\texttt{action}(t)$, is true for a given time $t$ tells us which action to perform at time $t$.

For sequential planning we have axioms to ensure that only one action is performed at each time.

$$action_1(t) \wedge action_2(t) \rightarrow action_1 = action_2$$

Parallel planning, which we will not discuss, allows more than one action to be performed concurrently.

We also add axioms to represent the correct operation of each action. We again use the `pickup` action from blocks world example to illustrate the idea.

**Preconditions** An action can only be performed if its preconditions hold.

$$\texttt{pickup}(x,t) \rightarrow \texttt{ontable}(x,t) \wedge \texttt{clear}(x,t) \wedge \texttt{handempty}(t)$$

**Effects** When an action is performed, it causes its effects.

$$\texttt{pickup}(x,t) \rightarrow$$
$$\neg\texttt{ontable}(x,t+1) \wedge \neg\texttt{clear}(x,t+1) \wedge \neg\texttt{handempty}(x,t+1) \wedge \texttt{holding}(x,t+1)$$

**Frame axioms** A predicate can only be changed by specified actions.

$$\texttt{pickup}(x,t) \vee \texttt{putdown}(x,t) \vee (\texttt{ontable}(x,t+1) \leftrightarrow \texttt{ontable}(x,t))$$

A model for the axioms that also satisfies, for example, $\texttt{init}(0)$ and $\texttt{goal}(7)$, would give us a seven step plan for achieving the goal. Instantiating the axioms for all blocks is unproblematic, since there are finitely many blocks. However, we cannot instantiate them for all times. So the procedure is to first instantiate for some number, $N$ of times. If we can solve the problem with the constraint $\texttt{goal}(m)$, for some $m < N$, then we have an $m$-step plan. If not we instantiate for more times and try to find a longer plan.

So for the blocks example, with three blocks, we have 16 fully parametrised fluents, as above. We also have 24 fully parametrised actions (9 versions of each action with two parameters, 3 of each with one parameter). to represent plans with 7 steps we need a logic with $(7 \times 24 + 8 \times 16)$ propositional letters[3].

# Appendix: Second Order Propositional Logic

In Propositional Logic statements are formed from propositional letters using boolean connectives; a model is just a truth valuation on the propositional letters.

In Predicate Logic, we add predicates, – unary, binary, $n$-ary – first-order predicates are interpreted as relations on, or properties of, individuals[4]. A model is based on a set of individuals; an $n$-ary predicate is interpreted as an $n$-ary relation on this set. Predicate Logic includes propositional logic – propositional letters are just nullary predicates.

Quantification introduces another dimension. In First-order Predicate Logic we quantify over individuals. In Second-order Predicate Logic we can quantify over predicates.

Second-order Propositional Logic, or Quantified Boolean Formulae (QBF), is what we get when we add second-order quantifiers to propositional logic. Formally, we add universal and existential quantifiers over propositions. Since, in the standard semantics, a proposition can only take two values, $\top$, or $\bot$, we can express these using conjunction and disjunction.

$$\forall A.\varphi(A) \equiv \varphi(\top) \wedge \varphi(\bot)$$
$$\exists A.\varphi(A) \equiv \varphi(\top) \vee \varphi(\bot)$$

For example, let $\varphi(B)$ be the propositional formula $A \rightarrow (B \wedge C)$. Then $\exists B.\varphi(B)$ is $(A \rightarrow (\top \wedge C)) \vee (A \rightarrow (\bot \wedge C))$, which is equivalent $(A \rightarrow C) \vee \neg A$, which, in turn, reduces to $A \rightarrow C$.

[3]Note that these counts are for naïve representations where each fluent and each time step are represented without further analysis. Practical planning systems may, and do, analyse the situation, for example by identifying invariants, to produce more compact propositional representations.

[4]Second-order predicates would be interpreted as properties of first order properties. An example: the property of being true of at least five individuals. We don't consider these here.